ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models

Stefan Mitsch **André Platzer**

July 2014
CMU-CS-14-121

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A conference version of this report has appeared at RV 2014 [22].
Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. In Borzoo Bonakdarpour and Scott A. Smolka, editors, Runtime Verification - 5th International Conference, RV 2014, Toronto, Canada, September 22-25, 2014. Proceedings, volume 8734 of LNCS, pages 199-214. Springer, 2014.

This material is based on research sponsored by DARPA under agreement number DARPA FA8750-12-2-0291. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Formal verification and validation play a crucial role in making cyber-physical systems (CPS) safe. Formal methods make strong guarantees about the system behavior if accurate models of the system can be obtained, including models of the controller and of the physical dynamics. In CPS models are essential; but any model we could possibly build necessarily deviates from the real world. If the real system fits to the model, its behavior is guaranteed to satisfy the correctness properties verified w.r.t. the model. Otherwise, all bets are off. This paper introduces ModelPlex, a method ensuring that verification results about models apply to CPS implementations. ModelPlex provides correctness guarantees for CPS executions at runtime: it combines offline verification of CPS models with runtime validation of system executions for compliance with the model. Model- Plex ensures that the verification results obtained for the model apply to the actual system runs by monitoring the behavior of the world for compliance with the model, assuming the system dynamics deviation is bounded. If, at some point, the observed behavior no longer complies with the model so that offline verification results no longer apply, ModelPlex initiates provably safe fallback actions. This paper, furthermore, develops a systematic technique to synthesize provably correct monitors automatically from CPS proofs in differential dynamic logic.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

Keywords: runtime verification, cyber-physical systems, hybrid systems, logic

Abstract

Formal verification and validation play a crucial role in making cyber-physical systems (CPS) safe. Formal methods make strong guarantees about the system behavior *if* accurate models of the system can be obtained, including models of the controller and of the physical dynamics. In CPS, models are essential; but any model we could possibly build necessarily deviates from the real world. If the real system fits to the model, its behavior is guaranteed to satisfy the correctness properties verified w.r.t. the model. Otherwise, all bets are off. This paper introduces ModelPlex, a method ensuring that verification results about models apply to CPS implementations. ModelPlex provides correctness guarantees for CPS executions at runtime: it combines offline verification of CPS models with runtime validation of system executions for compliance with the model. ModelPlex ensures that the verification results obtained for the model apply to the actual system runs by monitoring the behavior of the world for compliance with the model, assuming the system dynamics deviation is bounded. If, at some point, the observed behavior no longer complies with the model so that offline verification results no longer apply, ModelPlex initiates provably safe fallback actions. This paper, furthermore, develops a systematic technique to synthesize provably correct monitors automatically from CPS proofs in differential dynamic logic.

1 Introduction

Cyber-physical systems (CPS) span controllers and the relevant dynamics of the environment. Since safety is crucial for CPS, their models (e. g., hybrid system models [29]) need to be verified formally. Formal *verification* guarantees that a model is safe w.r.t. a safety property. The remaining task is to *validate* whether those models are adequate, so that the verification results transfer to the system implementation [16, 38]. This paper introduces ModelPlex, a method to *synthesize monitors by theorem proving*: it uses sound proof rules to formally verify that a model is safe and to synthesize provably correct monitors that validate compliance of system executions with that model.

System execution, however, provides many opportunities for surprising deviations from the model: faults may cause the system to function improperly [39], sensors may deliver uncertain values, actuators suffer from disturbance, or the formal verification may have assumed simpler ideal-world dynamics for tractability reasons or made unrealistically strong assumptions about the behavior of other agents in the environment. Simpler models are often better for real-time decisions and optimizations, because they make predictions feasible to compute at the required rate. The same phenomenon of simplicity for predictability is often exploited for the models in formal verification and validation. As a consequence, the *verification results obtained about models of a CPS only apply to the actual CPS at runtime to the extent that the system fits to the model*.

Validation, i. e., checking whether a CPS implementation fits to a model, is an interesting but difficult problem. Even more so, since CPS models are more difficult to analyze than ordinary (discrete) programs because of the physical plant, the environment, sensor inaccuracies, and actuator disturbance. In CPS, models are essential; but any model we could possibly build necessarily deviates from the real world. Still, good models are approximately right, i. e., within certain error margins.

In this paper, we settle for the question of *runtime model validation*, i. e. validating whether the model assumed for verification purposes is adequate for a *particular system execution* to ensure that the verification results apply *to the current execution*.[1] But we focus on *verifiably correct runtime validation* to ensure that verified properties of models provably apply, which is important for safety and certification [5].

If the observed system execution fits to the verified model, then this execution is safe according to the offline verification result about the model. If it does not fit, then the system is potentially unsafe because it no longer has an applicable safety proof, so we initiate a verified fail-safe action to avoid safety risks. Checking whether a system execution fits to a verified model includes checking that the actions chosen by the (unverified) controller implementation fit to *one of* the choices and requirements of the verified controller model. It also includes checking that the observed states can be explained by the plant model. The crucial questions are: How can a compliance monitor be synthesized that provably represents the verified model? How much safety margin does a system need to ensure that fail-safe actions are initiated early enough for the system to remain safe even if

[1] ModelPlex checks system execution w.r.t. a monitor specification, and thus, belongs to the field of runtime verification [16]. In this paper we use the term *runtime validation* in order to clearly convey the purpose of monitoring (i. e., runtime verification: monitor properties without offline verification; ModelPlex: monitor model adequacy to transfer offline verification results).

its behavior ceases to comply with the model?

The second question is related to feedback control and can only be answered when assuming constraints on the deviation of the real system dynamics from the plant model [33]. Otherwise, i. e., if the real system can be infinitely far off from the model, safety guarantees are impossible. By the sampling theorem in signal processing [37], such constraints further enable compliance monitoring solely on the basis of sample points instead of the unobservable intermediate states about which no sensor data exists.[2] This paper presents ModelPlex, a method to synthesize verifiably correct runtime validation monitors automatically. ModelPlex uses theorem proving with sound proof rules [29] to turn hybrid system models into monitors in a verifiably correct way. Upon noncompliance, ModelPlex initiates provably safe fail-safe actions. System-level challenges w.r.t. monitor implementation and violation cause diagnosis are discussed elsewhere [8, 19, 41].

2 Preliminaries: Differential Dynamic Logic

For hybrid systems verification we use *differential dynamic logic* dL [27, 29, 31], which has a notation for hybrid systems as *hybrid programs*. dL allows us to make statements that we want to be true for all runs of a hybrid program ($[\alpha]\phi$) or for at least one run ($\langle\alpha\rangle\phi$). Both constructs are necessary to derive safe monitors: we need $[\alpha]\phi$ proofs so that we can be sure all behavior of a model (including controllers) are safe; we need $\langle\alpha\rangle\phi$ proofs to find monitor specifications that detect whether or not system execution fits to the verified model. Table 1 summarizes the relevant syntax fragment of hybrid programs together with an informal semantics. The semantics $\rho(\alpha)$ of hybrid program α is a relation on initial and final states of running α (defined in [27, 30]). The set of dL formulas is generated by the following grammar ($\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

[2] When such constraints are not available, our method still generates verifiably correct *runtime tests*, which detect deviation from the model at the sampling points, just not between them. A fail-safe action will then lead to best-effort mitigation of safety risks (rather than guaranteed safety).

Table 1: Hybrid program representations of hybrid systems.

Statement	Effect
$\alpha;\ \beta$	sequential composition, first run hybrid program α, then hybrid program β
$\alpha \cup \beta$	nondeterministic choice, following either hybrid program α or β
α^*	nondeterministic repetition, repeats hybrid program α $n \geq 0$ times
$x := \theta$	assign value of term θ to variable x (discrete jump)
$x := *$	assign arbitrary real number to variable x
$?F$	check that a particular condition F holds, and abort if it does not
$\big(x_1' = \theta_1, \ldots, x_n' = \theta_n \ \&\ F\big)$	evolve x_i along differential equation system $x_i' = \theta_i$ restricted to maximum evolution domain F

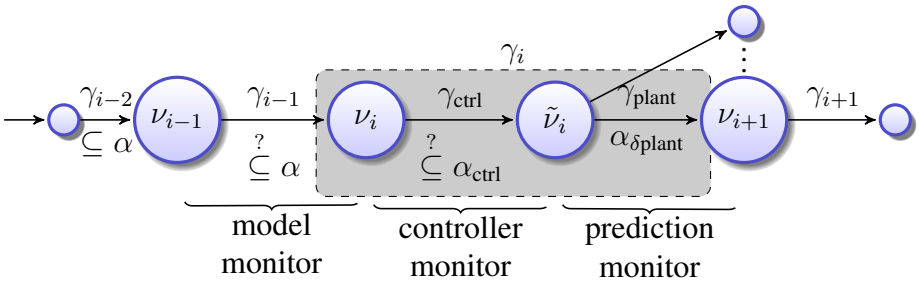

Figure 1: Use of ModelPlex monitors along a system execution

Differential dynamic logic comes with a verification technique to prove correctness properties of hybrid programs (cf. [31] for an overview of dL and KeYmaera).

3 ModelPlex Approach for Verified Runtime Validation

CPS are almost impossible to get right without sufficient attention to prior analysis, for instance by formal verification and formal validation techniques. We assume to be given a verified model of a CPS, i. e. formula (1) is proved valid,[3] for example using [27, 31].

$$\phi \rightarrow [\alpha^*]\psi \qquad \text{with invariant } \varphi \rightarrow [\alpha]\varphi \text{ s.t. } \phi \rightarrow \varphi \text{ and } \varphi \rightarrow \psi \tag{1}$$

Formula (1) expresses that all runs of the hybrid system α^*, which start in states that satisfy the precondition ϕ and repeat the model α arbitrarily many times, must end in states that satisfy the postcondition ψ. Formula (1) is proved using some form of induction, which shows that a loop invariant φ holds after every run of α if it was true before. The model α is a *hybrid system model* of a CPS, which means that it describes both the discrete control actions of the controllers in the system and the continuous physics of the plant and the system's environment.

The safety guarantees that we obtain by proving formula (1) about the model α^* transfer to the real system, *if* the actual CPS execution fits to α^*. Since we want to preserve safety properties, a CPS γ fits to a model α^*, *if* the CPS reaches at most those states that are reachable by the model, i. e., $\rho(\gamma) \subseteq \rho(\alpha^*)$. However, we do not know γ and therefore need to find a condition based on α^* that we can check at runtime to see if concrete runs of γ behave like α^*. Checking the postcondition ψ is not sufficient because, if ψ does not hold, the system is already unsafe. Checking the invariant φ is insufficient as well, because if φ does not hold the controller can no longer guarantee safety, even though the system may not yet be unsafe. But if we detect when a CPS is about to deviate from α^* *before* leaving φ, we can still switch to a fail-safe controller to avoid $\neg\psi$ from happening.

ModelPlex derives three kinds of monitors (model monitor, controller monitor, and prediction monitor, cf. Fig. 1). We check reachability between consecutive states in α, α_{ctrl}, and $\alpha_{\delta\text{plant}}$ by verifying states during execution against the corresponding monitor.

[3] We use differential dynamic logic (dL) and KeYmaera as a theorem prover to illustrate our concepts throughout this paper. The concept of ModelPlex is not predicated on the use of KeYmaera to prove (1). Other verification techniques could be used to establish validity of this formula. The flexibility of the underlying logic dL, its support for both $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, and its proof calculus, however, are exploited for systematically constructing monitors from proofs in the sequel.

Model monitor In each state ν_i we test the sample point ν_{i-1} from the previous execution γ_{i-1} for deviation from the single α, not α^* i. e., test $(\nu_{i-1}, \nu_i) \in \rho(\alpha)$. If violated, other verified properties may no longer hold for the system; the system, however, is still safe if a prediction monitor was satisfied on ν_{i-1}. Frequent violations indicate an inadequate model that should be revised to better reflect reality.

Controller monitor In intermediate state $\tilde{\nu}_i$ we test the current controller decisions of the implementation γ_{ctrl} for compliance with the model, i. e., test $(\nu_i, \tilde{\nu}_i) \in \rho(\alpha_{\text{ctrl}})$. Controller monitors are designed for switching between controllers similar to Simplex [36]. If violated, the commands from a fail-safe controller replace the current controller's decisions to ensure that no unsafe commands are ever actuated.

Prediction monitor In intermediate state $\tilde{\nu}_i$ we test the worst-case safety impact of the current controller decisions w.r.t. the predictions of a bounded deviation plant model $\alpha_{\delta\text{plant}}$, which has a tolerance around the model plant α_{plant}, i. e., check $\nu_{i+1} \models \varphi$ for all ν_{i+1} such that $(\tilde{\nu}_i, \nu_{i+1}) \in \rho(\alpha_{\delta\text{plant}})$. Note, that we simultaneously check all ν_{i+1} by checking $\tilde{\nu}_i$ for a characterizing condition of $\alpha_{\delta\text{plant}}$. If violated, the current control choice is not guaranteed to keep the system safe until the next control cycle and, thus, a fail-safe controller takes over.

The assumption for the prediction monitor is that the real execution is not arbitrarily far off the plant models used for safety verification, because otherwise guarantees can be neither made on unobservable intermediate states nor on safety of the future system evolution [33]. We propose separation of disturbance causes in the models: ideal plant models α_{plant} for correctness verification purposes, implementation deviation plant models $\alpha_{\delta\text{plant}}$ for monitoring purposes. We support any deviation model (e. g., piecewise constant disturbance, differential inclusion models of disturbance), as long as the deviation is bounded and differential invariants can be found. We further assume that monitor evaluations are at most some ε time units apart (e. g., along with a recurring controller execution). Note that disturbance in $\alpha_{\delta\text{plant}}$ is more manageable compared to α^*, because we can focus on single runs α instead of repetitions for monitoring.

3.1 Relation between States

We systematically derive a check that inspects states of the actual CPS to detect deviation from the model α^*. We first establish a notion of state recall and show that, when all previous state pairs complied with the model, compliance of the entire execution can be checked by checking the latest two states (ν_{i-1}, ν_i) (see App. A for proofs).

Definition 1 (State recall)**.** *We use V to denote the set of variables whose state we want to recall. We use $\Upsilon_V^- \equiv \bigwedge_{x \in V} x = x^-$ to express a characterization of the values of variables in a state prior to a run of α, where we always assume the fresh variables x^- to occur solely in Υ_V^-. The variables in x^- can be used to recall this state. Likewise, we use $\Upsilon_V^+ \equiv \bigwedge_{x \in V} x = x^+$ to characterize the posterior states and expect fresh x^+.*

With this notation the following lemma states that an interconnected sequence of α transitions forms a transition of α^*.

Lemma 1 (Loop prior and posterior state)**.** *Let α be a hybrid program and α^* be the program that repeats α arbitrarily many times. Assume that all consecutive pairs of states $(\nu_{i-1}, \nu_i) \in \rho(\alpha)$ of $n \in \mathbb{N}^+$ executions, whose valuations are recalled with $\Upsilon_V^i \equiv \bigwedge_{x \in V} x = x^i$ and Υ_V^{i-1} are plausible w.r.t. the model α, i. e., $\models \bigwedge_{1 \leq i \leq n} \left(\Upsilon_V^{i-1} \rightarrow \langle\alpha\rangle \Upsilon_V^i\right)$ with $\Upsilon_V^- = \Upsilon_V^0$ and $\Upsilon_V^+ = \Upsilon_V^n$. Then, the sequence of states originates from an α^* execution from Υ_V^0 to Υ_V^n, i. e., $\models \Upsilon_V^- \rightarrow \langle\alpha^*\rangle \Upsilon_V^+$.*

Lemma 1 enables us to check compliance with the model α^* up to the current state by checking reachability of a posterior state from a prior state on each execution of α (i. e., online monitoring [16], which is easier because the loop was eliminated). To find compliance checks systematically, we construct formula (2), which relates a prior state of a CPS to its posterior state through at least one path through the model α. [4]

$$\Upsilon_V^- \rightarrow \langle\alpha\rangle \Upsilon_V^+ \tag{2}$$

This formula is satisfied in a state ν, if there is at least one run of the model α starting in the state ν recalled by Υ_V^- and results in a state ω recalled using Υ_V^+. In other words, at least one path through α explains how the prior state ν got transformed into the posterior state ω. The dL formula (2) characterizes the state transition relation of the model α directly. Its violation witnesses compliance violation. Compliance at all intermediate states cannot be observed by real-world sensors, see Section 3.5.

In principle, formula (2) would be a monitor, because it relates a prior state to a posterior state through the model of a CPS; but the formula is hard if not impossible to evaluate at runtime, because it refers to a hybrid system α, which includes nondeterminism and differential equations. The basic observation is that any formula that is equivalent to (2) but conceptually easier to evaluate in a state would be a correct monitor. We use theorem proving for simplifying formula (2) into quantifier-free first-order real arithmetic form so that it can be evaluated efficiently at runtime. The resulting first-order real arithmetic formula can be easily implemented in a runtime monitor and executed along with the actual controller. A monitor is executable code that only returns true if the transition from the prior system state to the posterior state is compliant with the model. Thus, deviations from the model can be detected at runtime, so that appropriate fallback and mitigation strategies can be initiated.

Remark 1. *The complexity for evaluating an arithmetic formula over the reals for concrete numbers is linear in the formula size, as opposed to deciding the validity of such formulas, which is doubly exponential. Evaluating the same formula on floating point numbers is inexpensive, but may yield wrong results due to rounding errors; on exact rationals the bit-complexity can be non-negligible. We use interval arithmetic to obtain reliable results efficiently (cf. App. C).*

Example 1. *We will use a simple water tank as a running example to illustrate the concepts throughout this section. The water tank has a current level x and a maximum level m. The water tank controller, which runs at least every ε time units, nondeterministically chooses any flow f between a maximum outflow -1 and a maximum inflow $\frac{m-x}{\varepsilon}$. This water tank never overflows, as witnessed by a proof for the following dL formula.*

[4] Consecutive states for α^* mean before and after executions of α (i. e., $\alpha^{\downarrow};\alpha^{\downarrow};\alpha$, not within α).

$$\underbrace{0 \leq x \leq m \wedge \varepsilon > 0}_{\phi} \rightarrow \Big[\big(f := *;\ ?\big(-1 \leq f \leq \tfrac{m-x}{\varepsilon}\big);\ t := 0;\ (x' = f,\ t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon)\big)^* \Big] \overbrace{(0 \leq x \leq m)}^{\psi}$$

3.2 ModelPlex Monitor Synthesis

This section introduces the nature of ModelPlex monitor specifications, our approach to generate such specifications from hybrid system models, and how to turn those specifications into monitor code that can be executed at runtime along with the controller.

A ModelPlex specification corresponds to the dL formula (2). If the current state of a system does not satisfy a ModelPlex specification, some behavior that is not reflected in the model occurred (e. g., the wrong control action was taken, unanticipated dynamics in the environment occurred, sensor uncertainty led to unexpected values, or the system was applied outside the specified operating environment).

A *model monitor* χ_{m} checks that two consecutive states ν and ω can be explained by an execution of the model α, i. e., $(\nu, \omega) \in \rho(\alpha)$. In the sequel, $BV(\alpha)$ are bound variables in α, $FV(\psi)$ are free variables in ψ, Σ is the set of all variables, and $A \backslash B$ denotes the set of variables being in some set A but not in some other set B. Furthermore, we use $\nu|_A$ to denote ν projected onto the variables in A.

Theorem 1 (Model monitor correctness)**.** *Let α^* be provably safe, so $\models \phi \rightarrow [\alpha^*]\psi$. Let $V_m = BV(\alpha) \cup FV(\psi)$. Let $\nu_0, \nu_1, \nu_2, \nu_3 \ldots \in \mathbb{R}^n$ be a sequence of states, with $\nu_0 \models \phi$ and that agree on $\Sigma \backslash V_m$, i. e., $\nu_0|_{\Sigma \backslash V_m} = \nu_k|_{\Sigma \backslash V_m}$ for all k. We define $(\nu, \nu_{i+1}) \models \chi_m$ as χ_m evaluated in the state resulting from ν by interpreting x^+ as $\nu_{i+1}(x)$ for all $x \in V_m$, i. e., $\nu_{x^+}^{\nu_{i+1}(x)} \models \chi_m$. If $(\nu_i, \nu_{i+1}) \models \chi_m$ for all $i < n$ then we have $\nu_n \models \psi$ where*

$$\chi_m \equiv \big(\phi|_{const} \rightarrow \langle\alpha\rangle \Upsilon^+_{V_m}\big) \tag{3}$$

and $\phi|_{const}$ denotes the conditions of ϕ that involve only constants that do not change in α, i. e., $FV(\phi|_{const}) \cap BV(\alpha) = \emptyset$.

Our approach to generate monitor specifications from hybrid system models takes a verified dL formula (1) as input and produces a monitor χ_{m} in quantifier-free first-order form as output. The algorithm, listed in App. D, involves the following steps:

1. A dL formula (1) about a model α of the form $\phi \rightarrow [\alpha^*]\psi$ is turned into a specification conjecture (3) of the form $\phi|_{\mathrm{const}} \rightarrow \langle\alpha\rangle \Upsilon^+_{V_m}$.
2. Theorem proving on the specification conjecture (3) is applied until no further proof rules are applicable and only first-order real arithmetic formulas remain open.
3. The monitor specification χ_{m} is the conjunction of the unprovable first-order real arithmetic formulas from open sub-goals.

Generate the monitor conjecture. We map dℒ formula (1) syntactically to a specification conjecture of the form (3). By design, this conjecture will not be provable. But the unprovable branches of a proof attempt will reveal information that, had it been in the premises, would make (3) provable. Through $\Upsilon^+_{V_m}$, those unprovable conditions collect the relations of the posterior state of model α characterized by x^+ to the prior state x, i. e., the conditions are a representation of (2) in quantifier-free first-order real arithmetic.

Example 2. *The specification conjecture for the water tank model is given below. It is constructed from the model by removing the loop, flipping the modality, and formulating the specification requirement as a property, since we are interested in a relation between two consecutive states ν and ω (recalled by x^+, f^+ and t^+). Using theorem proving [34], we analyze the conjecture to reveal the actual monitor specification.*

$$\underbrace{\varepsilon > 0}_{\phi|_{const}} \rightarrow \Big\langle \begin{array}{l} f := *; ?\left(-1 \leq f \leq \frac{m-x}{\varepsilon}\right); \\ t := 0;\ (x' = f,\ t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon) \end{array} \Big\rangle \overbrace{(x = x^+ \wedge f = f^+ \wedge t = t^+)}^{\Upsilon^+_{V_m}}$$

Use theorem proving to analyze the specification conjecture. We use the proof rules of dℒ [27, 31] to analyze the specification conjecture χ_{m}. These proof rules syntactically decompose a hybrid model into easier-to-handle parts, which leads to sequents with first-order real arithmetic formulas towards the leaves of a proof. Using real arithmetic quantifier elimination we close sequents with logical tautologies, which do not need to be checked at runtime since they always evaluate to *true* for any input. The conjunction of the remaining open sequents is the monitor specification; it implies (2).

A complete sequence of proof rules applied to the monitor conjecture of the water tank is described in App. B. Most steps are simple when analyzing specification conjectures: sequential composition ($\langle;\rangle$), nondeterministic choice ($\langle\cup\rangle$), deterministic assignment ($\langle:=\rangle$) and logical connectives ($\wedge$r etc.) replace current facts with simpler ones or branch the proof (cf. rules in [27, 30]). Challenges arise from handling nondeterministic assignment and differential equations in hybrid programs.

Let us first consider nondeterministic assignment $x := *$. The proof rule for nondeterministic assignment ($\langle * \rangle$) results in a new existentially quantified variable. By sequent proof rule $\exists$r, this existentially quantified variable is instantiated with an arbitrary term θ, which is often a new logical variable that is implicitly existentially quantified [27]. Weakening (Wr) removes facts that are no longer necessary.

$$(\langle * \rangle)\ \frac{\exists X \langle x := X \rangle \phi}{\langle x := * \rangle \phi}\ {}^{1} \qquad (\exists\mathrm{r})\ \frac{\Gamma \vdash \phi(\theta), \exists x\, \phi(x), \Delta}{\Gamma \vdash \exists x\, \phi(x), \Delta}\ {}^{2} \qquad (\mathrm{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta}$$

[1] X is a new logical variable

[2] θ is an arbitrary term, often a new (existential) logical variable X.

Optimization 1 (Instantiation Trigger)**.** *If the variable is not changed in the remaining α, $x_i = x_i^+$ is in $\Upsilon^+_{V_m}$ and X is not bound in $\Upsilon^+_{V_m}$, then instantiate the existential quantifier by rule $\exists$r with the corresponding x_i^+ that is part of the specification conjecture (i. e., $\theta = x_i^+$), since subsequent proof steps are going to reveal $\theta = x_i^+$ anyway.*

Otherwise, we introduce a new logical variable, which may result in an existential quantifier in the monitor specification if no further constraints can be found later in the proof.

Example 3. *The corresponding steps in the water tank proof use $\langle * \rangle$ for the nondeterministic flow assignment ($f := *$) and $\exists r$ to instantiate the resulting existential quantifier $\exists F$ with a new logical variable F (plant is an abbreviation for $x' = f, t' = 1 \,\&\, 0 \leq x \wedge t \leq \varepsilon$). We show the proof without and with application of Opt. 1.*

$$\dfrac{\dfrac{\phi \vdash \langle f := F\rangle\langle ?{-1} \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle \mathit{plant}\rangle \Upsilon^+ \quad \text{w/o Opt. 1}}{\phi \vdash \exists F \langle f := F\rangle\langle ?{-1} \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle \mathit{plant}\rangle \Upsilon^+}\,{}^{\exists r,Wr}}{\phi \vdash \langle f := *; ?{-1} \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle \mathit{plant}\rangle \Upsilon^+}\,{}^{\langle * \rangle} \qquad \dfrac{\phi \vdash \langle f := f^+\rangle \langle ?{-1} \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle \mathit{plant}\rangle \Upsilon^+}{\dots}\,{}^{\exists r,Wr}$$

with Opt. 1 (anticipate $f = f^+$ from Υ^+)

Next, we handle differential equations. Even when we can solve the differential equation, existentially and universally quantified variables remain. Let us inspect the corresponding proof rule from the dL calculus [31].

$$(\langle' \rangle) \quad \frac{\exists T \geq 0 \left((\forall 0 \leq \tilde{t} \leq T \, \langle x := y(\tilde{t})\rangle H) \wedge \langle x := y(T)\rangle \phi \right)}{\langle x' = \theta \,\&\, H\rangle \phi}\,{}^{1} \qquad (\text{QE}) \quad \frac{\text{QE}(\phi)}{\phi}\,{}^{2}$$

[1] T and $\tilde{t}$ are fresh logical variables and $\langle x := y(T)\rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value

[2] iff $\phi \equiv \text{QE}(\phi)$, ϕ is a first-order real arithmetic formula, $\text{QE}(\phi)$ is an equivalent quantifier-free formula computable by [7]

For differential equations we have to prove that there exists a duration t, such that the differential equation stays within the evolution domain H throughout all intermediate times $\tilde{t}$ and the result satisfies ϕ at the end. At this point we have three options:

- we can instantiate the existential quantifier, if we know that the duration will be t^+;
- we can introduce a new logical variable, which is the generic case that always yields correct results, but may discover monitor specifications that are harder to evaluate;
- we can use quantifier elimination (QE) to obtain an equivalent quantifier-free result (a possible optimization could inspect the size of the resulting formula).

Example 4. *In the analysis of the water tank example, we solve the differential equation (see $\langle' \rangle$) and apply the substitutions $f := F$ and $t := 0$. In the next step (see $\exists r$,Wr), we instantiate the existential quantifier $\exists T$ with t^+ (i. e., we choose $T = t^+$ using Opt. 1 with the last conjunct) and use weakening right (Wr) to systematically get rid of the existential quantifier that would otherwise still be left around by rule $\exists r$. Finally, we use quantifier elimination (QE) to reveal an equivalent quantifier-free formula.*

The analysis of the specification conjecture finishes with collecting the open sequents from the proof to create the monitor specification $\chi_{\text{m}} \overset{\text{def}}{\equiv} \bigwedge(\textit{open sequent})$. The collected open sequents may

$$\dfrac{\dfrac{\dfrac{\phi \vdash F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0}{\phi \vdash \forall 0 \leq \tilde{t} \leq T\ (x + f^+\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon) \wedge F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ = t^+}\,\text{QE}}{\phi \vdash \exists T \geq 0((\forall 0 \leq \tilde{t} \leq T\ (x + f^+\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon)) \wedge F = f^+ \wedge (x^+ = x + FT \wedge t^+ = T))}\,\exists\text{r},\text{Wr}}{\phi \vdash \langle f := F; t := 0\rangle\langle\{x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon\}\rangle \Upsilon^+}\,\langle'\rangle$$

include new logical variables and new (Skolem) function symbols that were introduced for nondeterministic assignments and differential equations when handling existential or universal quantifiers. We use the invertible quantifier rule i$\exists$ to re-introduce existential quantifiers for the new logical variables (universal quantifiers for function symbols, see [27] for calculus details). Often, the now quantified logical variables are discovered to be equal to one of the post-state variables later in the proof, because those variables did not change in the model after the assignment. If this is the case, we can use proof rule $\exists\sigma$ to further simplify the monitor specification by substituting the corresponding logical variable x with its equal term θ.

$$(\text{i}\exists)\quad \frac{\Gamma \vdash \exists X\ (\bigwedge_i (\Phi_i \vdash \Psi_i)), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \cdots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}\ {}^{1} \qquad (\exists\sigma)\quad \frac{\phi(\theta)}{\exists x\ (x = \theta \wedge \phi(x))}\ {}^{2}$$

[1] Among all open branches, free logical variable X only occurs in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$

[2] Logical variable x does not appear in term θ

Example 5. *The two open sequents of Examples 3 and 4 use a new logical variable F for the nondeterministic flow assignment $f := *$. After further steps in the proof, the assumptions reveal additional information $F = f^+$. Thus, we re-introduce the existential quantifier over all the open branches (i$\exists$) and substitute f^+ for F ($\exists\sigma$). The sole open sequent of this proof attempt is the monitor specification χ_m of the water tank model.*

$$\dfrac{\dfrac{\phi \vdash -1 \leq f^+ \leq \frac{m-x}{\varepsilon} \wedge x^+ = x + f^+t^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge f^+t^+ + x \geq 0}{\phi \vdash \exists F(-1 \leq F \leq \frac{m-x}{\varepsilon} \wedge F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0)}\,\exists\sigma}{\phi \vdash -1 \leq F \leq \frac{m-x}{\varepsilon} \qquad \phi \vdash F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0}\,\text{i}\exists$$

3.3 Controller Monitor Synthesis

A *controller monitor* χ_c checks that two consecutive states ν and ω are reachable with one controller execution α_{ctrl}, i. e., $(\nu, \omega) \in \rho(\alpha_{\text{ctrl}})$ with $V_c = BV(\alpha_{\text{ctrl}}) \cup FV(\psi)$. We systematically derive controller monitors from formulas $\phi|_{\text{const}} \rightarrow \langle\alpha_{\text{ctrl}}\rangle\Upsilon^+_{V_c}$. A controller monitor can be used to initiate controller switching similar to Simplex [36].

Theorem 2 (Controller monitor correctness)**.** *Let α of the canonical form $\alpha_{ctrl}; \alpha_{plant}$. Assume $\models \phi \rightarrow [\alpha^*]\psi$ has been proven with invariant φ as in (1). Let $\nu \models \phi|_{const} \wedge \varphi$, as checked by χ_m (Theorem 1). Furthermore, let $\tilde{\nu}$ be a post-controller state. If $(\nu, \tilde{\nu}) \models \chi_c$ with $\chi_c \equiv \phi|_{const} \rightarrow \langle\alpha_{ctrl}\rangle\Upsilon^+_{V_c}$ then we have that $(\nu, \tilde{\nu}) \in \rho(\alpha_{ctrl})$ and $\tilde{\nu} \models \varphi$.*

3.4 Monitoring in the Presence of Expected Uncertainty and Disturbance

Up to now we considered exact ideal-world models. But real-world clocks drift, sensors measure with some uncertainty, and actuators are subject to disturbance. This makes the exact models safe but too conservative, which means that monitors for exact models are likely to fall back to a fail-safe controller rather often. In this section we discuss how we find ModelPlex specifications so that the safety property (1) and the monitor specification become more robust to expected uncertainty and disturbance. That way, only unexpected deviations beyond those captured in the normal operational uncertainty and disturbance of α^* cause the monitor to initiate fail-safe actions.

In dL, we can, for example, use nondeterministic assignment from an interval to model sensor uncertainty and piecewise constant actuator disturbance (e. g., as in [24]), or differential inequalities for actuator disturbance (e. g., as in [35]). Such models include nondeterminism about sensed values in the controller model and often need more complex physics models than differential equations with polynomial solutions.

Example 6. *We incorporate clock drift, sensor uncertainty and actuator disturbance into the water tank model to express expected deviation. The measured level x_s is within a known sensor uncertainty u of the real level x (i.e. $x_s \in [x-u, x+u]$). We use differential inequalities to model clock drift and actuator disturbance. The clock, which wakes the controller, is slower than the real time by at most a time drift of c; it can be arbitrarily fast. The water flow disturbance is at most d, but the water tank is allowed to drain arbitrarily fast (even leaks when the pump is on). To illustrate different modeling possibilities, we use additive clock drift and multiplicative actuator disturbance.*

$$0 \le x \le m \land \varepsilon > 0 \land\ c < 1 \land 0 \le u \land 0 < d$$
$$\to \Big[\big(x_s := *; ?\,(x-u \le x_s \le x+u)\,;\ f := *; ?\,\big(-1 \le f \le \tfrac{m-x_s-u}{d\varepsilon}(1-c)\big);$$
$$t := 0;\ \{x' \le fd,\ 1-c \le t' \,\&\, x \ge 0 \land t \le \varepsilon\}\big)^* \Big](0 \le x \le m)$$

We analyze Example 6 in the same way as the previous examples, with the crucial exception of the differential inequalities. We cannot use the proof rule $\langle' \rangle$ to analyze this model, because differential inequalities do not have polynomial solutions. Instead, we use the DR and DE proof rules of dL [28, 29] to turn differential inequalities into a differential-algebraic constraint form that lets us proceed with the proof. Rule DE turns a differential inequality $x' \le \theta$ into a quantified differential equation $\exists \tilde{d}(x' = \tilde{d}\ \&\ \tilde{d} \le \theta)$ with an equivalent differential-algebraic constraint. Rule DR turns a differential-algebraic constraint $\mathscr{E}$ into another differential-algebraic constraint $\mathscr{D}$, which implies $\mathscr{E}$, written $\mathscr{D} \to \mathscr{E}$, as defined in [28] (cf. App. B.1 for an example).

$$\text{(DR)}\quad \frac{\mathscr{D} \to \mathscr{E} \quad \langle \mathscr{D} \rangle \phi}{\langle \mathscr{E} \rangle \phi}\ {}^{1} \qquad \text{(DE)}\quad \frac{\begin{array}{r}\forall X(\exists \tilde{d}(X = \tilde{d} \land \tilde{d} \le \theta \land H) \to X \le \theta \land H) \\ \langle \exists \tilde{d}(x' = \tilde{d}\,\&\,\tilde{d} \le \theta \land H) \rangle \phi\end{array}}{\langle x' \le \theta\,\&\,H \rangle \phi}\ {}^{2}$$

[1] differential refinement: differential-algebraic constraints $\mathscr{D}, \mathscr{E}$ have the same changed variables

[2] differential inequality elimination: special case of DR, which rephrases the differential inequalities $\le$ as differential-algebraic constraints (accordingly for other or mixed inequalities systems).

Currently, for finding model monitors our prototype tool solves differential equations by the proof rule $\langle' \rangle$. Thus, it finds model monitor specifications for differential algebraic equations with

polynomial solutions and for differential algebraic inequalities, which can be refined into solvable differential algebraic equations as in Example 6. For prediction monitors (discussed in Section 3.5) we use dℒ techniques for finding differential variants and invariants, differential cuts [28], and differential auxiliaries [32] to handle differential equations and inequalities *without polynomial solutions*.

3.5 Monitoring Compliance Guarantees for Unobservable Intermediate States

With controller monitors, non-compliance of a controller implementation w.r.t. the modeled controller can be detected right away. With model monitors, non-compliance of the actual system dynamics w.r.t. the modeled dynamics can be detected when they first occur. We switch to a fail-safe action, which is verified using standard techniques, in both non-compliance cases. The crucial question is: can such a method always guarantee safety? The answer is linked to the image computation problem in model checking (i. e., approximation of states reachable from a current state), which is known to be not semi-decidable by numerical evaluation at points; approximation with uniform error is only possible if a bound is known for the continuous derivatives [33]. This implies that we need additional assumptions about the deviation between the actual and the modeled continuous dynamics to guarantee compliance for unobservable intermediate states. Unbounded deviation from the model between sample points just is unsafe, no matter how hard a controller tries. Hence, worst-case bounds capture how well reality is reflected in the model.

We derive a prediction monitor to check whether a current control decision will be able to keep the system safe for time ε even if the actual continuous dynamics deviate from the model. A prediction monitor checks the current state, because all previous states are ensured by a model monitor and subsequent states are then safe by (1).

Definition 2 (ε-bounded plant with disturbance δ)**.** *Let α_{plant} be a model of the form $x' = \theta \,\&\, H$. An* ε-bounded plant with disturbance δ*, written $\alpha_{\delta plant}$, is a plant model of the form*

$$x_0 := 0;\ \ (f(\theta,\delta) \leq x' \leq g(\theta,\delta) \,\&\, H \wedge x_0 \leq \varepsilon)$$

for some f, g with fresh variable $\varepsilon > 0$ and assuming $x_0' = 1$. We say that disturbance δ is constant *if $x \notin \delta$; it is* additive *if $f(\theta,\delta) = \theta - \delta$ and $g(\theta,\delta) = \theta + \delta$.*

Theorem 3 (Prediction monitor correctness)**.** *Let α^* be provably safe, i. e., $\models \phi \rightarrow [\alpha^*]\psi$ has been proved using invariant φ as in (1). Let $V_p = BV(\alpha) \cup FV([\alpha]\varphi)$. Let $\nu \models \phi|_{const} \wedge \varphi$, as checked by χ_m from Theorem 1. Further assume $\tilde{\nu}$ such that $(\nu,\tilde{\nu}) \in \rho(\alpha_{ctrl})$, as checked by χ_c from Theorem 2. If $(\nu,\tilde{\nu}) \models \chi_p$ with $\chi_p \equiv (\phi|_{const} \wedge \varphi) \rightarrow \langle \alpha_{ctrl} \rangle (\Upsilon^+_{V_p} \wedge [\alpha_{\delta plant}]\varphi)$, then we have for all $(\tilde{\nu},\omega) \in \rho(\alpha_{\delta plant})$ that $\omega \models \varphi$.*

Remark 2. *By adding a controller execution $\langle \alpha_{ctrl} \rangle$ prior to the disturbed plant model, we synthesize prediction monitors that take the actual controller decisions into account. For safety purposes, we could just as well use a monitor definition without controller $\chi_p \equiv (\phi|_{const} \wedge \varphi) \rightarrow [\alpha_{\delta plant}]\varphi$. But doing so results in a conservative monitor, which has to keep the CPS safe without knowledge of the actual controller decision.*

Table 2: Monitor complexity case studies

	Case Study	Model		Monitor				Time/Mem.
		dim.	proof size (branches)	dim.	steps (open seq.) w/ Opt. 1	steps (open seq.) auto	proof steps (branches)	size	([s]/[MB])
χ_m	Water tank	5	38 (4)	3	16 (2)	20 (2)	64 (5)	32	2.2 / 45.3
	Cruise control [18]	11	969 (124)	7	127 (13)	597 (21)	19514 (1058)	1111	42.8 / 54.9
	Speed limit [23]	9	410 (30)	6	487 (32)	5016 (126)	64311 (2294)	19850	239.1 / 49.7
χ_c	Water tank	5	38 (4)	1	12 (2)	14 (2)	40 (3)	20	1.3 / 24.6
	Cruise control [18]	11	969 (124)	7	83 (13)	518 (106)	5840 (676)	84	16.4 / –[i]
	Robot [24]	14	3350 (225)	11	94 (10)	1210 (196)	26166 (2854)	121	39.2 / –[1]
	ETCS safety [35]	16	193 (10)	13	162 (13)	359 (37)	16770 (869)	153	14.8 / –[1]
χ_p	Water tank	8	80 (6)	1	135 (4)	N/A	307 (12)	43	16.7 / 47.7[ii]

http://www.cs.cmu.edu/~smitsch/resource/modelplex_study.zip

[i] No memory consumption recorded [ii] Not automated, replaying the proof containing manual steps

3.6 Decidability and Computability

One useful characteristic of ModelPlex beyond soundness is that monitor synthesis is computable, which yields a synthesis algorithm, and that the correctness of those synthesized monitors w.r.t. their specification is decidable, cf. Theorem 4.

Theorem 4 (Monitor correctness is decidable and monitor synthesis computable)**.** *We assume canonical models of the form* $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ *without nested loops, with solvable differential equations in* α_{plant} *and disturbed plants* $\alpha_{\delta plant}$ *with constant additive disturbance* δ *(see Def. 2). Then, monitor correctness is decidable, i. e., the formulas* $\chi_m \rightarrow \langle\alpha\rangle\Upsilon_V^+$, $\chi_c \rightarrow \langle\alpha_{ctrl}\rangle\Upsilon_V^+$, *and* $\chi_p \rightarrow \langle\alpha\rangle(\Upsilon_V^+ \wedge [\alpha_{\delta plant}]\phi)$ *are decidable. Also, monitor synthesis is computable, i. e., the functions* $synth_m : \langle\alpha\rangle\Upsilon_V^+ \mapsto \chi_m$, $synth_c : \langle\alpha_{ctrl}\rangle\Upsilon_V^+ \mapsto \chi_c$, *and* $synth_p : \langle\alpha\rangle(\Upsilon_V^+ \wedge [\alpha_{\delta plant}]\phi) \mapsto \chi_p$ *are computable.*

4 Evaluation

We developed a software prototype, integrated into our modeling tool Sphinx [25], to automate many of the described steps. The prototype generates χ_m, χ_c, and χ_p conjectures from hybrid programs, collects open sequents, and interacts with KeYmaera [34].

To evaluate our method, we created monitors for prior case studies of non-deterministic hybrid models of autonomous cars, train control systems, and robots (adaptive cruise control [18], intelligent speed adaptation [23], the European train control system [35], and ground robot collision avoidance [24]). Table 2 summarizes the evaluation. For the model, we list the dimension in terms of the number of function symbols and state variables, and the size of the safety proof (i. e., number of proof steps and branches). For the monitor, we list the dimension of the monitor conjecture in terms of the number of variables, compare the number of steps and open sequents when

deriving the monitor using manual proof steps to apply Opt. 1 and fully automated w/o Opt. 1, and the number of steps in the monitor correctness proof. Finally, we list the monitor size in terms of arithmetic, comparison, and logical operators in the monitor formula. Although the number of steps and open sequents differ significantly between manual interaction for Opt. 1 and fully automated synthesis, the synthesized monitors are logically equivalent. But applying Opt. 1 usually results in structurally simpler monitors, because the conjunction over a smaller number of open sequents (cf. Table 2) can still be simplified automatically. The model monitors for cruise control and speed limit control are significantly larger than the other monitors, because their size already prevents automated simplification by Mathematica. As future work, KeYmaera will be adapted to allow user-defined tactics in order to apply Opt. 1 automatically. The last column lists duration and memory consumption for automated monitor synthesis in KeYmaera without Opt. 1. Finding ModelPlex and PredictPlex monitors is quite challenging, in comparison to finding Simplex monitors, because of the additional plant model with mostly non-trivial differential equations. We further simulated monitors in Mathematica. The simulation results are discussed in App. E.

5 Related Work

Runtime verification and monitoring for finite state discrete systems has received significant attention (e. g., [9, 14, 21]). Other approaches monitor continuous-time signals (e. g., [10, 26]). We focus on hybrid systems models of CPS to combine both.

Several tools for formal verification of hybrid systems are actively developed (e. g., SpaceEx [12], dReal [13], extended NuSMV/MathSat [6]). For monitor synthesis, however, ModelPlex crucially needs the rewriting capabilities and flexibility of (nested) $[\alpha]$ and $\langle\alpha\rangle$ modalities in $\mathsf{d}\mathcal{L}$ [29] and KeYmaera [34]; it is thus an interesting question for future work if other tools could be adapted to ModelPlex.

Runtime verification is the problem of checking whether or not a trace produced by a program satisfies a particular formula (cf. [16]). In [40], a method for runtime verification of LTL formulas on abstractions of concrete traces of a flight data recorder is presented. The RV system for Java programs [20] predicts execution traces from actual traces to find concurrency errors offline (e. g., race conditions) even if the actual trace did not exhibit the error. We, instead, use prediction on the basis of disturbed plant models for *hybrid systems* at runtime to ensure safety for future behavior of the system and switch to a fail-safe fallback controller if necessary. Adaptive runtime verification [4] uses state estimation to reduce monitoring overhead by sampling while still maintaining accuracy with Hidden Markov Models, or more recently, particle filtering [15] to fill the sampling gaps. The authors present interesting ideas for managing the overhead of runtime monitoring, which could be beneficial to transfer into the hybrid systems world. The approach, however, focuses purely on the discrete part of CPS.

The Simplex architecture [36] (and related approaches, e. g., [1, 3, 17]) is a control system principle to switch between a highly reliable and an experimental controller at runtime. Highly reliable control modules are assumed to be verified with some other approach. Simplex focuses on switching when timing faults or violation of controller specification occur. Our method complements Simplex in that *(i)* it checks whether or not the current system execution fits the entire

model, not just the controller; *(ii)* it systematically derives provably correct monitors for hybrid systems; *(iii)* it uses prediction to guarantee safety for future behavior of the system.

Further approaches with interesting insights on combined verification and monitor/controller synthesis for discrete systems include, for instance, [2, 11].

Although the related approaches based on offline verification derive monitors and switching conditions from models, none of them validates whether or not the model is adequate for the current execution. Thus, they are vulnerable to deviation between the real world and the model. In summary, this paper addresses safety at runtime as follows:

- Unlike [36], who focus on timing faults and specification violations, we propose a systematic principle to derive monitors that react to any deviation from the model.
- Unlike [4, 15, 17, 20], who focus on the discrete aspects of CPS, we use hybrid system models with differential equations to address controller and plant.
- Unlike [17, 36], who assume that fail-safe controllers have been verified with some other approach and do not synthesize code, we can use the same technical approach (dℒ) for verifying controllers and synthesizing provably correct monitors.
- ModelPlex combines the leight-weight monitors and runtime compliance of online runtime verification with the design time analysis of offline verification.
- ModelPlex synthesizes provably correct monitors, certified by a theorem prover
- To the best of our knowledge, our approach is the first to guarantee that verification results about a hybrid systems model transfer to a particular execution of the system by verified runtime validation. We detect deviation from the verified model when it first occurs and, given bounds, can guarantee safety with fail-safe fallback. Other approaches (e. g., [3, 17, 36]) assume the system perfectly complies with the model.

6 Conclusion

ModelPlex is a principle to build and verify high-assurance controllers for safety-critical computerized systems that interact physically with their environment. It guarantees that verification results about CPS models transfer to the real system by safeguarding against deviations from the verified model. Monitors created by ModelPlex are provably correct and check at runtime whether or not the actual behavior of a CPS complies with the verified model and its assumptions. Upon noncompliance, ModelPlex initiates fail-safe fallback strategies. In order to initiate those strategies early enough, ModelPlex uses prediction on the basis of disturbed plant models to check safety for the next control cycle. This way, ModelPlex ensures that verification results about a model of a CPS transfer to the actual system behavior at runtime.

Future research directions include extending ModelPlex with advanced dℒ proof rules for differential equations [31], so that differential equations without polynomial solutions, as we currently handle for prediction monitor synthesis, can be handled for model monitor synthesis as well. An

interesting question for certification purposes is end-to-end verification from the model to the final machine code.

Acknowledgments.

We thank the anonymous reviewers of the conference version [22] for their careful reading and their helpful comments.

References

[1] Anthony M. Aiello, John F. Berryman, Jonathan R. Grohs, and John D. Schierman. Run-time assurance for advanced flight-critical control systems. In *AIAA Guidance, Nav. and Control Conf.* AIAA, 2010. doi:10.2514/6.2010-8041.

[2] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *FMCAD*, pages 1–17. IEEE, 2013.

[3] Stanley Bak, Ashley Greer, and Sayan Mitra. Hybrid cyberphysical system verification with Simplex using discrete abstractions. In Marco Caccamo, editor, *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 143–152. IEEE Computer Society, 2010. ISBN 978-0-7695-4001-6.

[4] Ezio Bartocci, Radu Grosu, Atul Karmarkar, Scott A. Smolka, Scott D. Stoller, Erez Zadok, and Justin Seyster. Adaptive runtime verification. In Shaz Qadeer and Serdar Tasiran, editors, *RV*, volume 7687 of *LNCS*, pages 168–182. Springer, 2012. ISBN 978-3-642-35631-5, 978-3-642-35632-2.

[5] Jan Olaf Blech, Yliès Falcone, and Klaus Becker. Towards certified runtime verification. In Toshiaki Aoki and Kenji Taguchi, editors, *ICFEM*, volume 7635 of *LNCS*, pages 494–509. Springer, 2012. ISBN 978-3-642-34280-6.

[6] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. SMT-based scenario verification for hybrid systems. *Formal Methods in System Design*, 42(1):46–66, 2013.

[7] George E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, 1991.

[8] Matthew J. Daigle, Indranil Roychoudhury, Gautam Biswas, Xenofon D. Koutsoukos, Ann Patterson-Hine, and Scott Poll. A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 40(5):917–931, 2010.

[9] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: Runtime monitoring of synchronous systems. In *TIME*, pages 166–174. IEEE Computer Society, 2005. ISBN 0-7695-2370-6.

[10] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 264–279. Springer, 2013. ISBN 978-3-642-39798-1.

[11] Rüdiger Ehlers and Bernd Finkbeiner. Monitoring realizability. In Sarfraz Khurshid and Koushik Sen, editors, *RV*, volume 7186 of *LNCS*, pages 427–441. Springer, 2011.

[12] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011. ISBN 978-3-642-22109-5.

[13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013. ISBN 978-3-642-38573-5.

[14] Klaus Havelund and Grigore Rosu. Efficient monitoring of safety properties. *STTT*, 6(2): 158–173, 2004.

[15] Kenan Kalajdzic, Ezio Bartocci, Scott A. Smolka, Scott D. Stoller, and Radu Grosu. Runtime verification with particle filtering. In Axel Legay and Saddek Bensalem, editors, *RV*, volume 8174 of *LNCS*. Springer, 2013. ISBN 978-3-642-40786-4.

[16] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[17] Xue Liu, Qixin Wang, Sathish Gopalakrishnan, Wenbo He, Lui Sha, Hui Ding, and Kihwal Lee. ORTEGA: An efficient and flexible online fault tolerance architecture for real-time control systems. *IEEE Trans. Industrial Informatics*, 4(4):213–224, 2008.

[18] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM*, volume 6664 of *LNCS*. Springer, 2011. doi:10.1007/978-3-642-21437-0_6.

[19] Sheila A. McIlraith, Gautam Biswas, Dan Clancy, and Vineet Gupta. Hybrid systems diagnosis. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *LNCS*, pages 282–295. Springer, 2000. ISBN 3-540-67259-1.

[20] Patrick O'Neil Meredith and Grigore Rosu. Runtime verification with the RV system. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *RV*, volume 6418 of *LNCS*, pages 136–152. Springer, 2010. ISBN 978-3-642-16611-2.

[21] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Rosu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.

[22] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *RV*, volume 8734 of *LNCS*, pages 199–214. Springer, 2014.

[23] Stefan Mitsch, Sarah M. Loos, and André Platzer. Towards formal verification of freeway traffic control. In Chenyang Lu, editor, *ICCPS*, pages 171–180. IEEE, 2012. ISBN 978-0-7695-4695-7. doi:10.1109/ICCPS.2012.25.

[24] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In Paul Newman, Dieter Fox, and David Hsu, editors, *Robotics: Science and Systems*, 2013. ISBN 978-981-07-3937-9. URL http://www.roboticsproceedings.org/rss09/p14.pdf.

[25] Stefan Mitsch, Grant Olney Passmore, and André Platzer. Collaborative verification-driven engineering of hybrid systems. *Mathematics in Computer Science*, 8(1):71–97, 2014. doi:10.1007/s11786-014-0176-y.

[26] Dejan Nickovic and Oded Maler. AMT: A property-based monitoring tool for analog systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, LNCS, pages 304–319. Springer, 2007. ISBN 978-3-540-75453-4.

[27] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. ISSN 0168-7433. doi:10.1007/s10817-008-9103-8.

[28] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. doi:10.1093/logcom/exn070. Advance Access published on November 18, 2008.

[29] André Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010. ISBN 978-3-642-14508-7. doi:10.1007/978-3-642-14509-4.

[30] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. ISBN 978-1-4673-2263-8. doi:10.1109/LICS.2012.13.

[31] André Platzer. The complete proof theory of hybrid systems. In *LICS*. IEEE, 2012. ISBN 978-1-4673-2263-8. doi:10.1109/LICS.2012.64.

[32] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. doi:10.2168/LMCS-8(4:16)2012.

[33] André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors, *HSCC*, LNCS. Springer, 2007. ISBN 978-3-540-71492-7. doi:10.1007/978-3-540-71493-4_37.

[34] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*. Springer, 2008. ISBN 978-3-540-71069-1. doi:10.1007/978-3-540-71070-7_15.

[35] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*. Springer, 2009. doi:10.1007/978-3-642-10373-5_13.

[36] Danbing Seto, Bruce Krogh, Lui Sha, and Alongkrit Chutinan. The Simplex architecture for safe online control system upgrades. In *American Control Conference*, pages 3504–3508, 1998. doi:10.1109/ACC.1998.703255.

[37] C.E. Shannon. Communication in the presence of noise. *Proc. of the IRE*, 37(1):10–21, 1949. ISSN 0096-8390. doi:10.1109/JRPROC.1949.232969.

[38] Ashok N. Srivastava and Johann Schumann. Software health management: a necessity for safety critical systems. *ISSE*, 9(4):219–233, 2013.

[39] D. Wang, M. Yu, C. B. Low, and S. Arogeti. *Model-based Health Monitoring of Hybrid Systems*. Springer, 2013. ISBN 978-1-4614-7369-5. doi:10.1007/978-1-4614-7369-5.

[40] Shaohui Wang, Anaheed Ayoub, Oleg Sokolsky, and Insup Lee. Runtime verification of traces under recording uncertainty. In Sarfraz Khurshid and Koushik Sen, editors, *RV*, LNCS, pages 442–456. Springer, 2011.

[41] Feng Zhao, Xenofon D. Koutsoukos, Horst W. Haussecker, James Reich, and Patrick Cheung. Monitoring and fault diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1225–1240, 2005.

A Proofs

A.1 Formal Semantics of dℒ

ModelPlex bases on a reachability relation semantics instead of trace semantics [29], since it is easier to handle and suffices for checking at sample points.

The semantics of dℒ, as defined in [27], is a Kripke semantics in which states of the Kripke model are states of the hybrid system. Let $\mathbb{R}$ denote the set of real numbers. A state is a map $\nu : V \to \mathbb{R}$; the set of all states is denoted by Sta. We write $\nu \models \phi$ if formula ϕ is true at state ν (Def. 4). Likewise, $[\![\theta]\!]_\nu$ denotes the real value of term θ at state ν. The semantics of HP α is captured by the state transitions that are possible by running α. For continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous flow respecting the differential equation and invariant region. That is, there is a continuous transition along $x' = \theta \,\&\, H$ from state ν to state ω, if there is a solution of the differential equation $x' = \theta$ that starts in state ν and ends in ω and that always remains within the region H during its evolution.

Definition 3 (Transition semantics of hybrid programs)**.** *The* transition relation ρ *specifies which state ω is reachable from a state ν by operations of α. It is defined as follows.*

1. $(\nu, \omega) \in \rho(x := \theta)$ *iff* $[\![z]\!]_\nu = [\![z]\!]_\omega$ *f.a.* $z \neq x$ *and* $[\![x]\!]_\omega = [\![\theta]\!]_\nu$.
2. $(\nu, \omega) \in \rho(x := *)$ *iff* $[\![z]\!]_\nu = [\![z]\!]_\omega$ *f.a.* $z \neq x$.
3. $(\nu, \omega) \in \rho(?\phi)$ *iff* $\nu = \omega$ *and* $\nu \models \phi$.
4. $(\nu, \omega) \in \rho(x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H)$ *iff for some* $r \geq 0$*, there is a (*flow*) function* $\varphi{:}[0, r] \to \mathrm{Sta}$ *with* $\varphi(0) = \nu, \varphi(r) = \omega$*, such that for each time* $\zeta \in [0, r]$*: (i) The differential equation holds, i.e.,* $\frac{\mathsf{d}\,[\![x_i]\!]_{\varphi(t)}}{\mathsf{d}t}(\zeta) = [\![\theta_i]\!]_{\varphi(\zeta)}$ *for each* x_i*. (ii) For other variables* $y \notin \{x_1, \ldots, x_n\}$ *the value remains constant, i.e.,* $[\![y]\!]_{\varphi(\zeta)} = [\![y]\!]_{\varphi(0)}$*. (iii) The invariant is always respected, i.e.,* $\varphi(\zeta) \models H$.
5. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
6. $\rho(\alpha; \beta) = \{(\nu, \omega) : (\nu, z) \in \rho(\alpha), (z, \omega) \in \rho(\beta)$ *for a state* $z\}$
7. $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ *where* $\alpha^{i+1} \hat{=} (\alpha; \alpha^i)$ *and* $\alpha^0 \hat{=} ?true$.

Definition 4 (Interpretation of dℒ formulas)**.** *The interpretation* $\models$ *of a* dℒ *formula with respect to state ν is defined as follows.*

1. $\nu \models \theta_1 \sim \theta_2$ *iff* $[\![\theta_1]\!]_\nu \sim [\![\theta_2]\!]_\nu$ *for* $\sim \in \{=, \leq, <, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ *iff* $\nu \models \phi$ *and* $\nu \models \psi$*, accordingly for* $\neg, \vee, \rightarrow, \leftrightarrow$
3. $\nu \models \forall x\, \phi$ *iff* $\omega \models \phi$ *for all ω that agree with ν except for the value of x*

4. $\nu \models \exists x\, \phi$ *iff* $\omega \models \phi$ *for some* ω *that agrees with* ν *except for the value of* x

5. $\nu \models [\alpha]\phi$ *iff* $\omega \models \phi\ \forall \omega$ *with* $(\nu, \omega) \in \rho(\alpha)$

6. $\nu \models \langle\alpha\rangle\phi$ *iff* $\omega \models \phi\ \exists \omega$ *with* $(\nu, \omega) \in \rho(\alpha)$

We write $\models \phi$ *to denote that* ϕ *is* valid*, i. e., that* $\nu \models \phi\ \forall \nu$.

A.2 Soundness

We recall Lemma 1.

Lemma 1 (Loop prior and posterior state)**.** *Let* α *be a hybrid program and* α^* *be the program that repeats* α *arbitrarily many times. Assume that all consecutive pairs of states* $(\nu_{i-1}, \nu_i) \in \rho(\alpha)$ *of* $n \in \mathbb{N}^+$ *executions, whose valuations are recalled with* $\Upsilon_V^i \equiv \bigwedge_{x \in V} x = x^i$ *and* Υ_V^{i-1} *are plausible w.r.t. the model* α*, i. e.,* $\models \bigwedge_{1 \leq i \leq n} \left(\Upsilon_V^{i-1} \rightarrow \langle\alpha\rangle \Upsilon_V^i\right)$ *with* $\Upsilon_V^- = \Upsilon_V^0$ *and* $\Upsilon_V^+ = \Upsilon_V^n$*. Then, the sequence of states originates from an* α^* *execution from* Υ_V^0 *to* Υ_V^n*, i. e.,* $\models \Upsilon_V^- \rightarrow \langle\alpha^*\rangle \Upsilon_V^+$.

Proof. Follows from the transition semantics of α^*: $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ where $\alpha^{i+1} \hat{=} (\alpha; \alpha^i)$ and $\alpha^0 \hat{=} ?true$. □

We recall Theorem 1.

Theorem 1 (Model monitor correctness)**.** *Let* α^* *be provably safe, so* $\models \phi \rightarrow [\alpha^*]\psi$*. Let* $V_m = BV(\alpha) \cup FV(\psi)$*. Let* $\nu_0, \nu_1, \nu_2, \nu_3 \ldots \in \mathbb{R}^n$ *be a sequence of states, with* $\nu_0 \models \phi$ *and that agree on* $\Sigma \backslash V_m$*, i. e.,* $\nu_0|_{\Sigma \backslash V_m} = \nu_k|_{\Sigma \backslash V_m}$ *for all* k*. We define* $(\nu, \nu_{i+1}) \models \chi_m$ *as* χ_m *evaluated in the state resulting from* ν *by interpreting* x^+ *as* $\nu_{i+1}(x)$ *for all* $x \in V_m$*, i. e.,* $\nu_{x^+}^{\nu_{i+1}(x)} \models \chi_m$*. If* $(\nu_i, \nu_{i+1}) \models \chi_m$ *for all* $i < n$ *then we have* $\nu_n \models \psi$ *where*

$$\chi_m \equiv \left(\phi|_{const} \rightarrow \langle\alpha\rangle \Upsilon_{V_m}^+\right) \tag{3}$$

and $\phi|_{const}$ *denotes the conditions of* ϕ *that involve only constants that do not change in* α*, i. e.,* $FV(\phi|_{const}) \cap BV(\alpha) = \emptyset$.

Proof. By induction over n. If $n = 0$ then $(\nu_0, \nu_0) \in \rho(\alpha^*)$ trivially by definition of ρ and $\models \phi \rightarrow [\alpha^*]\psi$ implies $\nu_0 \models \psi$. For $n > 0$ assume $(\nu_0, \nu_n) \in \rho(\alpha^*)$ and $(\nu_n, \nu_{n+1}) \models \langle\alpha\rangle \bigwedge_{x \in V_m} x = x^+$. Then there exists μ such that $(\nu_{n\ x^+}^{\ \nu_{n+1}(x)}, \mu) \in \rho(\alpha)$ and the two states agree on all variables except the ones modified by α, i. e., $\nu_{n\ x^+}^{\ \nu_{n+1}(x)}|_{\Sigma \backslash BV(\alpha)} = \mu|_{\Sigma \backslash BV(\alpha)}$. Thus, $\mu \models \Upsilon_{V_m}^+$, i. e., $\mu \models \bigwedge_{x \in V_m} x = x^+$, which in turn yields $\mu(x) = \mu(x^+) = \nu_{n\ x^+}^{\ \nu_{n+1}(x)}(x^+) = \nu_{n+1}(x)$ (in other words, $\mu|_{V_m} = \nu_{n+1}|_{V_m}$). Since also $\nu_n|_{\Sigma \backslash V_m} = \nu_{n+1}|_{\Sigma \backslash V_m}$ we get $\mu = \nu_{n+1}$ and $(\nu_n, \nu_{n+1}) \in \rho(\alpha)$. Hence $(\nu_0, \nu_{n+1}) \in \rho(\alpha^*)$ because by induction hypothesis $(\nu_0, \nu_n) \in \rho(\alpha^*)$ and we conclude $\nu_{n+1} \models \psi$ by assumption $\models \phi \rightarrow [\alpha^*]\psi$ using $\nu_0 \models \phi$.

□

We recall Theorem 2.

Theorem 2 (Controller monitor correctness)**.** *Let α of the canonical form $\alpha_{ctrl};\alpha_{plant}$. Assume $\models \phi \rightarrow [\alpha^*]\psi$ has been proven with invariant φ as in (1). Let $\nu \models \phi|_{const} \wedge \varphi$, as checked by χ_m (Theorem 1). Furthermore, let $\tilde{\nu}$ be a post-controller state. If $(\nu,\tilde{\nu}) \models \chi_c$ with $\chi_c \equiv \phi|_{const} \rightarrow \langle\alpha_{ctrl}\rangle\Upsilon^+_{V_c}$ then we have that $(\nu,\tilde{\nu}) \in \rho(\alpha_{ctrl})$ and $\tilde{\nu} \models \varphi$.*

Proof. Consider a state $\nu \models \phi|_{\text{const}} \wedge \varphi$. Assume $(\nu,\tilde{\nu}) \models \chi_{\text{c}}$, i. e., $\nu^{\tilde{\nu}(x)}_{x^+} \models \chi_{\text{c}}$. Then there exists μ such that $(\nu^{\tilde{\nu}(x)}_{x^+},\mu) \in \rho(\alpha_{\text{ctrl}})$ and the two states agree on all variables except the ones modified by α_{ctrl}, i. e., $\nu^{\tilde{\nu}(x)}_{x^+}|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})} = \mu|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})}$. Thus, $\mu \models \Upsilon^+_{V_c}$, i. e., $\mu \models \bigwedge_{x\in V_c} x = x^+$, which in turn yields $\mu(x) = \mu(x^+) = \nu^{\tilde{\nu}(x)}_{x^+}(x^+) = \tilde{\nu}(x)$ (in other words, $\mu|_{V_c} = \tilde{\nu}|_{V_c}$). Since also $\mu|_{\Sigma\setminus V_c} = \tilde{\nu}|_{\Sigma\setminus V_c}$ we get $\mu = \tilde{\nu}$ and $(\nu,\tilde{\nu}) \in \rho(\alpha_{\text{ctrl}})$. Then we have $\tilde{\nu} \models \varphi$ because by assumption $\varphi \rightarrow [\alpha_{\text{ctrl}};\alpha_{\text{plant}}]\varphi$ and $\rho(\alpha_{\text{plant}})$ is reflexive as ODE can evolve for time 0. □

We recall Theorem 3.

Theorem 3 (Prediction monitor correctness)**.** *Let α^* be provably safe, i. e., $\models \phi \rightarrow [\alpha^*]\psi$ has been proved using invariant φ as in (1). Let $V_p = BV(\alpha) \cup FV([\alpha]\varphi)$. Let $\nu \models \phi|_{const} \wedge \varphi$, as checked by χ_m from Theorem 1. Further assume $\tilde{\nu}$ such that $(\nu,\tilde{\nu}) \in \rho(\alpha_{ctrl})$, as checked by χ_c from Theorem 2. If $(\nu,\tilde{\nu}) \models \chi_p$ with $\chi_p \equiv (\phi|_{const} \wedge \varphi) \rightarrow \langle\alpha_{ctrl}\rangle(\Upsilon^+_{V_p} \wedge [\alpha_{\delta plant}]\varphi)$, then we have for all $(\tilde{\nu},\omega) \in \rho(\alpha_{\delta plant})$ that $\omega \models \varphi$.*

Proof. Consider a state ν such that $\nu \models \phi|_{\text{const}} \wedge \varphi$. Let $\tilde{\nu}$ be some state such that $(\nu,\tilde{\nu}) \in \rho(\alpha_{\text{ctrl}})$. Then we have $\tilde{\nu} \models \varphi$ because by assumption $\varphi \rightarrow [\alpha_{\text{ctrl}};\ \alpha_{\text{plant}}]\varphi$ and $\rho(\alpha_{\text{plant}})$ is reflexive as ODE can evolve for time 0. Furthermore $\tilde{\nu} \models \phi|_{\text{const}}$ since $\nu|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})} = \tilde{\nu}|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})}$ and $FV(\phi|_{\text{const}}) \cap BV(\alpha_{\text{ctrl}}) = \emptyset$. Assume $(\nu,\tilde{\nu}) \models \chi_{\text{p}}$, i. e., $\nu^{\tilde{\nu}(x)}_{x^+} \models \chi_{\text{p}}$. Then there exists μ such that $\mu \models \Upsilon^+_{V_p} \wedge [\alpha_{\delta\text{plant}}]\varphi$ with $(\nu^{\tilde{\nu}(x)}_{x^+},\mu) \in \rho(\alpha_{\text{ctrl}})$ and the two states agree on all variables except the ones modified by α_{ctrl}, i. e., $\nu^{\tilde{\nu}(x)}_{x^+}|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})} = \mu|_{\Sigma\setminus BV(\alpha_{\text{ctrl}})}$. Thus, $\mu(x) = \mu(x^+) = \nu^{\tilde{\nu}(x)}_{x^+}(x^+) = \tilde{\nu}(x)$. (in other words, $\mu|_{V_p} = \tilde{\nu}|_{V_p}$). However, from χ_{p} we know that $\mu \models [\alpha_{\delta\text{plant}}]\varphi$. Thus, by the coincidence lemma [29, Lemma 2.6] $\tilde{\nu} \models [\alpha_{\delta\text{plant}}]\varphi$ since $FV([\alpha_{\delta\text{plant}}]\varphi) \subseteq V_p$ and hence we have $\omega \models \varphi$ for all $(\tilde{\nu},\omega) \in \rho(\alpha_{\delta\text{plant}})$. □

Observe that this is also true for all intermediate times $\zeta \in [0,\omega(t)]$ by the transition semantics of differential equations, where $\omega(t) \leq \varepsilon$ because $\alpha_{\delta\text{plant}}$ is bounded by ε.

A.3 Decidability and Computability

From Lemma 1 it follows that online monitoring [16] (i. e., monitoring the last two consecutive states) is permissible. So, ModelPlex turns questions $[\alpha^*]\phi$ and $\langle\alpha^*\rangle\phi$ into $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, respectively. For decidability, we first consider canonical hybrid programs α of the form $\alpha \equiv \alpha_{\text{ctrl}};\alpha_{\text{plant}}$ where α_{ctrl} and α_{plant} are free of further nested loops.

We split Theorem 4 (decidability and computability) into Theorem 5 (decidability) and Theorem 6 (computability) and prove them separately. To handle differential inequalities in $\mathsf{d}\mathcal{L}$ formulas of the form $[\alpha_{\delta\text{plant}}]\phi$, the subsequent proofs additionally assume the rules for handling differential-algebraic equations in the $\mathsf{d}\mathcal{L}$ calculus [29].

Theorem 5 (Monitor correctness is decidable)**.** *Monitor correctness is decidable for canonical models of the form* $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ *without nested loops, with solvable differential equations in* α_{plant} *and disturbed plants* $\alpha_{\delta plant}$ *with constant additive disturbance* δ*, i. e.,* $\chi_m \to \langle\alpha\rangle\Upsilon_V^+$*,* $\chi_c \to \langle\alpha_{ctrl}\rangle\Upsilon_V^+$*, and* $\chi_p \to \langle\alpha\rangle(\Upsilon_V^+ \wedge [\alpha_{\delta plant}]\phi)$ *are decidable.*

Proof. From relative decidability of dL [31, Theorem 11] we know that sentences of dL (i. e., dL formulas without free variables) are decidable relative to an oracle for discrete loop invariants/variants and continuous differential invariants/variants. Since neither α_{ctrl} nor α_{plant} contain nested loops, we manage without an oracle for loop invariants/variants. Further, since the differential equation systems in α_{plant} are solvable, we have an effective oracle for differential invariants/variants. Let $Cl_\forall(\phi)$ denote the universal closure of dL formula ϕ (i. e., $Cl_\forall(\phi) \equiv \forall_{z\in \text{FV}(\phi)} z.\phi$). Note that when $\models F$ then also $\models Cl_\forall(F)$ by a standard argument.

Model monitor $\chi_{\text{m}} \to \langle\alpha\rangle\Upsilon_V^+$: Follows from relative decidability of dL [31, Theorem 11], because $Cl_\forall(\chi_{\text{m}} \to \langle\alpha\rangle\Upsilon_V^+)$ contains no free variables.

Controller monitor $\chi_{\text{c}} \to \langle\alpha_{\text{ctrl}}\rangle\Upsilon_V^+$: Follows from relative decidability of dL [31, Theorem 11], because $Cl_\forall(\chi_{\text{c}} \to \langle\alpha_{\text{ctrl}}\rangle\Upsilon_V^+)$ contains no free variables.

Prediction monitor $\chi_{\text{p}} \to \langle\alpha_{\text{ctrl}}\rangle(\Upsilon_V^+ \wedge [\alpha_{\delta\text{plant}}]\phi)$: Decidability for α_{ctrl} follows from case $\chi_{\text{c}} \to \langle\alpha_{\text{ctrl}}\rangle\Upsilon_V^+$ (controller monitor) above. It remains to show decidability of $\chi_{\text{p}} \to \langle\alpha_{\text{ctrl}}\rangle[\alpha_{\delta\text{plant}}]\phi$, which by decidability of the controller monitor is $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [\alpha_{\delta\text{plant}}]\phi$. Since the disturbance δ in $\alpha_{\delta\text{plant}}$ is constant additive and the differential equations in α_{plant} are solvable, we have the disturbance functions $f(\theta,\delta)$ and $g(\theta,\delta)$ applied to the solution as an oracle[5] for differential invariants (i. e., the differential invariant is a pipe around the solution without disturbance). Specifically, to show $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [\alpha_{\delta\text{plant}}]\phi$ by Def. 2 we have to show $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [x_0 := 0;\ \{\theta - \delta \le x' \le \theta + \delta \,\&\, H \wedge x_0 \le \varepsilon\}]\phi$. We proceed with only $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [x_0 := 0;\ \{x' \le \theta + \delta \,\&\, H \wedge x_0 \le \varepsilon\}]\phi$ since the case $\theta - \delta \le x'$ follows in a similar manner. By definition of $\alpha_{\delta\text{plant}}$ we know $0 \le x_0$, and hence continue with $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [\{x' \le \theta + \delta \,\&\, H \wedge 0 \le x_0 \le \varepsilon\}]\phi$ by differential cut $0 \le x_0$. Using the differential cut rule [29], we further supply the oracle $\text{sol}_x + \delta x_0$, where sol_x denotes the solution of $x' = \theta$ in α_{plant} and δx_0 the solution for the disturbance since δ is constant additive. This leads to two proof obligations:

Prove oracle $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [x' \le \theta + \delta \,\&\, 0 \le x_0 \le \varepsilon]x \le \text{sol}_x + \delta x_0$, which by rule differential invariant [29] is valid if we can show $0 \le x_0 \le \varepsilon \to x' \le \text{sol}_x' + (\delta x_0)'$ where the primed variables are replaced with the respective right-hand side of the differential equation system. From Def. 2 we know that $x_0' = 1$ and $\delta' = 0$ and since sol_x is the solution of $x' = \theta$ in α_{plant} we further know that $\text{sol}_x' = \theta$; hence we have to show $0 \le x_0 \le \varepsilon \to \theta + \delta \le \theta + \delta$, which is trivially true.

Use oracle $(\chi_{\text{p}} \wedge \Upsilon_V^+) \to [x' \le \theta + \delta \,\&\, H \wedge 0 \le x_0 \le \varepsilon \wedge x \le \text{sol}_x + \delta x_0]\phi$, which by rule differential weaken [29] is valid if we can show

$$(\chi_{\text{p}} \wedge \Upsilon_V^+) \to \forall^\alpha\, ((H \wedge 0 \le x_0 \le \varepsilon \wedge x \le \text{sol}_x + \delta x_0) \to \phi)$$

[5] By design, the disturbed plant $\alpha_{\delta\text{plant}}$ also includes a clock x_0, so the oracle additionally includes the trivial differential invariant $x_0 \ge 0$.

where $\forall^\alpha$ denotes the universal closure w.r.t. x, i. e., $\forall x$. But, if χ_p is a correct monitor, this is provable by quantifier elimination. Furthermore, we cannot get a better result than differential weaken, because the evolution domain constraint contains the oracle's answer for the differential equation system, which characterizes exactly the reachable set of the differential equation system.

We conclude that the oracle is proven correct and its usage is decidable.

□

For computability, we start with a theoretical proof on the basis of decidability, before we give a constructive proof, which is more useful in practice.

Theorem 6 (Monitor synthesis is computable)**.** *Synthesis of χ_m, χ_c, and χ_p monitors is computable for canonical models of the form $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ without nested loops, with solvable differential equations in α_{plant} and plants $\alpha_{\delta plant}$ with constant additive disturbance δ, i. e., $synth_m : \langle\alpha\rangle\Upsilon_V^+ \mapsto \chi_m$, $synth_c : \langle\alpha_{ctrl}\rangle\Upsilon_V^+ \mapsto \chi_c$, and $synth_p : \langle\alpha\rangle(\Upsilon_V^+ \wedge [\alpha_{\delta plant}]\phi) \mapsto \chi_p$ are computable.*

Proof. Follows immediately from Theorem 5 with recursive enumeration of monitors. □

We give a constructive proof of Theorem 6. The proof is based on the observation that, except for loop and differential invariants/variants, rule application in the dL calculus is deterministic: from [29, Theorem 2.4] we know that, relative to an oracle for first-order invariants and variants, the dL calculus gives a semidecision-procedure for dL formulas with differential equations having first-order definable flows.

Proof. For the sake of a contradiction, suppose that monitor synthesis stopped with some open sequent not being a first-order quantifier-free formula. Then, by [29, Theorem 2.4] the open sequent either contains a hybrid program with nondeterministic repetition or a differential equation at top level, or it is not quantifier-free. But this contradicts our assumption that both α_ctrl and α_plant are free from loops and that the differential equations are solvable and disturbance is constant, in which case for

Model monitor synthesis $\chi_\mathbf{m}$: the solution rule $\langle' \rangle$ would make progress, because the differential equations in α_plant are solvable; and for

Prediction monitor synthesis $\chi_\mathbf{p}$: the disturbance functions $f(\theta, \delta)$ and $g(\theta, \delta)$ applied to the solution provide differential invariants (see proof of Theorem 5) so that the differential cut rule, the differential invariant rule, and the differential weakening rule [29] would make progress.

In the case of the open sequent not being quantifier-free, the quantifier elimination rule QE would be applicable and turn the formula including quantifiers into an equivalent quantifier-free formula. Hence, the open sequent neither contains nondeterministic repetition, nor a differential equation, nor a quantifier. Thus we conclude that the open sequent is a first-order quantifier-free formula. □

B Water Tank Monitor Specification Conjecture Analysis

Proof 1 shows a complete sequence of proof rules applied to the water tank specification conjecture of Example 2 on page 7, with $\phi \equiv \varepsilon > 0$ and $\Upsilon^+ \equiv x = x^+ \wedge f = f^+ \wedge t = t^+$.

B.1 Monitoring in the Presence of Expected Uncertainty and Disturbance

Example 7. *We start at the point where we have to handle the differential inequalities. First, we eliminate the differential inequalities by rephrasing them as differential-algebraic constraints in step (DE). Then, we refine by instantiating the existential quantifiers with the worst-case evolution in step (DR). The resulting differential equation has polynomial solutions and, thus, we can use $\langle' \rangle$ and proceed with the proof as before.*

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\dots}}{\phi \vdash \forall X \forall T(\exists \tilde{d} \exists \tilde{t}(X = \tilde{d} \wedge T = \tilde{t} \wedge \tilde{d} \leq fd \wedge 1 - c \leq \tilde{t} \wedge 0 \leq x \wedge t \leq \varepsilon \rightarrow X \leq F \wedge 1 - c \leq T \wedge 0 \leq \tilde{d} \wedge \tilde{t} \leq \varepsilon)} \quad \dfrac{\dfrac{*}{\dots}}{\phi \vdash \forall X \forall T(X = Fd \wedge T = 1 - c \wedge 0 \leq x \wedge t \leq \varepsilon \rightarrow \exists \tilde{d} \exists \tilde{t}(X = \tilde{d} \wedge \tilde{d} \leq Fd \wedge T = \tilde{t} \wedge 1 - c \leq T \wedge 0 \leq x \wedge t \leq \varepsilon)}}{\phi \vdash \exists T \geq 0 \big((\forall 0 \leq \tilde{t} \leq T (x + dFT \geq 0 \wedge \tilde{t}(1 - c) \leq \varepsilon)) \wedge F = f^+ \wedge x + dFT = x^+ \wedge X_s = x_s^+ \wedge T(1 - c) = t^+ \big)} (\chi_{m1})}{\phi \vdash \langle f := F; x_s := X_s; t := 0 \rangle \langle x' = fd, t' = 1 - c \,\&\, 0 \leq x \wedge t \leq \varepsilon \rangle \Upsilon^+} \langle' \rangle}{\phi \vdash \langle f := F; x_s := X_s; t := 0 \rangle \langle \exists \tilde{d} \exists \tilde{t}(x' = \tilde{d}, t' = \tilde{t} \,\&\, \tilde{d} \leq fd \wedge 1 - c \leq \tilde{t} \wedge 0 \leq x \wedge t \leq \varepsilon \rangle \Upsilon^+} \text{DR}}{\phi \vdash \langle f := F; x_s := X_s; t := 0 \rangle \langle x' \leq fd, 1 - c \leq t' \,\&\, 0 \leq x \wedge t \leq \varepsilon \rangle \Upsilon^+} \text{DE}$$

As expected, we get a more permissive monitor specification. One conjunct of the monitor specification is shown in (χ_{m1}). Such a monitor specification says that there exists a real flow F, a real time T, and a real level X_s, such that the measured flow f^+, the clock t^+, and the measured level x^+ can be explained with the model.

C Monitor Synthesis and Fallback Controller Design

C.1 Design-By-Contract Monitoring

Preconditions, postconditions and invariants are crucial conditions in CPS design. Monitors for these conditions can check *(i)* whether or not it is safe to start a particular controller (i. e., check that the precondition of a controller is satisfied), *(ii)* whether or not a controller complies with its specification (i. e., check that a controller delivers set values that satisfy its postcondition), and *(iii)* whether or not the system is still within its safety bounds (i. e., check that the loop invariant of α^* is satisfied).

Precondition and postcondition monitors are useful to decide whether or not it is safe to invoke a controller in the current state, and whether or not to trust a controller output. An invariant monitor

$$(\wedge r)\ \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \qquad (\text{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \qquad (\text{QE})\ \frac{\text{QE}(\phi)}{\phi}\ {}^{1}$$

$$(\langle ; \rangle)\ \frac{\langle \alpha \rangle \langle \beta \rangle \phi}{\langle \alpha ; \beta \rangle \phi} \qquad (\langle ? \rangle)\ \frac{H \wedge \psi}{\langle ?H \rangle \psi} \qquad (\langle := \rangle)\ \frac{\phi_x^\theta}{\langle x := \theta \rangle \phi} \qquad (\langle * \rangle)\ \frac{\exists X \langle x := X \rangle \phi}{\langle x := * \rangle \phi}\ {}^{2}$$

$$(\langle ' \rangle)\ \frac{\exists t {\geq} 0\ \big((\forall 0 {\leq} \tilde{t} {\leq} t\ \langle x := y(\tilde{t}) \rangle H) \wedge \langle x := y(t) \rangle \phi \big)}{\langle x' = \theta \,\&\, H \rangle \phi}\ {}^{3} \qquad (\exists r)\ \frac{\Gamma \vdash \phi(\theta), \exists x\, \phi(x), \Delta}{\Gamma \vdash \exists x\, \phi(x), \Delta}\ {}^{4}$$

$$(\text{i}\exists)\ \frac{\Gamma \vdash \exists X \left(\bigwedge_i (\Phi_i \vdash \Psi_i) \right), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \cdots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}\ {}^{5} \qquad (\exists \sigma)\ \frac{\phi_x^\theta}{\exists x\, (x = \theta \wedge \phi(x))}$$

[1] iff $\phi \equiv \text{QE}(\phi)$, ϕ is a first-order real arithmetic formula, QE(ϕ) is a quantifier-free formula
[2] X is a new logical variable
[3] t and $\tilde{t}$ are fresh logical variables and $\langle x := y(t) \rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value.
[4] θ is an arbitrary term, often a new (existential) logical variable X.
[5] Among all open branches, free logical variable X only occurs in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$

$$\phi \vdash -1 \leq f^+ \leq \tfrac{m-x}{\varepsilon} \wedge x^+ = x + f^+ t^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge f^+ t^+ + x \geq 0$$

$$\exists\sigma\ \overline{}$$

$$\phi \vdash \exists F \big(-1 \leq F \leq \tfrac{m-x}{\varepsilon} \wedge F = f^+ \wedge x^+ = x + F t^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge F t^+ + x \geq 0\big)$$

$$\overline{}$$

$$\phi \vdash F = f^+ \wedge x^+ = x + F t^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge F t^+ + x \geq 0$$

$$\text{QE}\ \overline{}$$

$$\phi \vdash \forall 0 {\leq} \tilde{t} {\leq} t^+\ (x + F\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon) \wedge F = f^+ \wedge x^+ = x + F t^+ \wedge t^+ = t^+$$

$$\exists r, \text{Wr}\ \overline{}$$

$$\phi \vdash \exists T {\geq} 0 \big((\forall 0 {\leq} \tilde{t} {\leq} T\ (x + F\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon)) \wedge (F = f^+ \wedge x^+ = x + FT \wedge t^+ = T) \big)$$

$$\langle ' \rangle\ \overline{}$$

$$\phi \vdash \langle f := F; t := 0 \rangle \langle x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon \rangle \Upsilon^+$$

$$\langle ; \rangle, \langle := \rangle\ \overline{}$$

$$\phi \vdash \langle f := F \rangle \langle t := 0; (x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon) \rangle \Upsilon^+$$

$$\overline{}$$

$$\text{i}\exists\ \frac{\phi \vdash -1 \leq F \leq \frac{m-x}{\varepsilon} \qquad \phi \vdash \langle f := F \rangle \langle \mathit{plant} \rangle \Upsilon^+}{\phi \vdash \langle f := F \rangle -1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle \mathit{plant} \rangle \Upsilon^+}\ \wedge r$$

$$\langle ? \rangle\ \overline{}$$

$$\phi \vdash \langle f := F \rangle \langle ? -1 \leq f \leq \tfrac{m-x}{\varepsilon} \rangle \langle \mathit{plant} \rangle \Upsilon^+$$

$$\exists r, \text{Wr}\ \overline{}$$

$$\phi \vdash \exists F \langle f := F \rangle \langle ? -1 \leq f \leq \tfrac{m-x}{\varepsilon} \rangle \langle \mathit{plant} \rangle \Upsilon^+$$

$$\langle * \rangle\ \overline{}$$

$$\phi \vdash \langle f := *; ? -1 \leq f \leq \tfrac{m-x}{\varepsilon} \rangle \langle \mathit{plant} \rangle \Upsilon^+$$

$$\langle ; \rangle\ \overline{}$$

$$\phi \vdash \langle f := *; ? -1 \leq f \leq \tfrac{m-x}{\varepsilon}; \mathit{plant} \rangle \Upsilon^+$$

Proof 1: Analysis of the water tank monitor specification conjecture (*plant* is an abbreviation for $x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon$)

of a CPS α^* captures the main assumptions that have to be true throughout system execution. When an invariant monitor is unsatisfied, it may no longer be safe to run the CPS; a fail-safe controller

can act as a mitigation strategy.

Design-by-contract monitors are useful to monitor specific design decisions, which are explicitly marked in the model. Our approach systematically creates monitors for a complete specification of the behavior of the model.

C.2 Monitor Synthesis

Once we found a model monitor, controller monitor, or prediction monitor specification, we want to turn it into an actual monitor implementation (e. g., in C). The main challenge is to reliably transfer the monitor specification, which is evaluated on $\mathbb{R}$, into executable code that uses floating point representations. We use the interval arithmetic library Apron to represent each real arithmetic value with an interval of a pair of floating point numbers. The interval reliably contains the real.

For certification purposes one still has to argue for the correctness of the actual machine code of the synthesized monitor. This entails that the transformation from the monitor specification as a first-order formula into actual code that evaluates the formula must be formally verified. If the synthesized code is still a high-level language, a certified compiler, e. g., CompCert[6], can be used to produce machine code. Such a comprehensive proof chain suitable for certification is part of our ongoing research.

C.3 Designing for a Fail-Safe Fallback Controller

When we design a system for a fail-safe fallback controller $ctrl_{\text{safe}}$, it is important to know within which bounds the fail-safe controller can still keep our CPS safe, and which design limits we want a controller implementation to obey. The invariant of a CPS with the fail-safe fallback controller describes the safety bounds. When we start the fail-safe fallback controller $ctrl_{\text{safe}}$ in a state where its invariant G is satisfied, it will guarantee to keep the CPS in a state that satisfies the safety property ψ.

So, to safely operate an experimental controller $ctrl_{\text{exp}}$, we want a monitor that informs us when the experimental controller can no longer guarantee the invariant of the fail-safe controller or when it is about to violate the design limits.

A design for a CPS with a fail-safe fallback controller, therefore, involves proving two properties. First, we prove that the fail-safe controller $ctrl_{\text{safe}}$ ensures the safety property ψ as in formula (4) below. This property is only provable if we discover an invariant G for the CPS with the fail-safe controller. Then we use G as the safety condition for generating a prediction monitor.

$$\phi \rightarrow [(ctrl_{\text{safe}};\ plant)^*@\text{inv}(G)]\psi \tag{4}$$

With this generic structure in mind, we can design for a fallback controller invoked by a model monitor χ_{m}, controller monitor χ_{c}, or prediction monitor χ_{p}. Upon violation of either χ_{m}, χ_{c}, or χ_{p} by the actual system execution, the set values of a fail-safe controller are used instead.

[6] http://compcert.inria.fr/

D Monitor Synthesis Algorithm

Algorithm 1 lists the ModelPlex specification conjecture analysis algorithm, which turns a specification conjecture into an actual monitor. The algorithm takes a hybrid system model α, a set of variables $\mathcal{V}$ that we want to monitor[7], and an initial condition ϕ including constraints on the variables not changed in α.

Algorithm 1: ModelPlex monitor synthesis

input : A hybrid program α, a set of variables $\mathcal{V} \subseteq BV(\alpha)$, an initial condition ϕ such that $\models \phi \rightarrow [\alpha^*]\psi$.
output: A monitor χ_m such that $\models \chi_\text{m} \equiv \phi|_\text{const} \rightarrow \langle\alpha\rangle\Upsilon^+$.
begin
  $S \longleftarrow \emptyset$
  $\Upsilon^+ \longleftarrow \bigwedge_{x\in\mathcal{V}} x = x^+$ with fresh variables x_i^+ `// Monitor conjecture`
  $G \longleftarrow \{\vdash \phi|_\text{const} \rightarrow \langle\alpha\rangle\Upsilon^+\}$
1 **while** $G \neq \emptyset$ **do** `// Analyze monitor conjecture`
    foreach $g \in G$ **do**
      $G \longleftarrow G - \{g\}$
      if g *is first-order* **then**
        if $\not\models g$ **then** $S \longleftarrow S \cup \{g\}$
      else
        $\tilde{g} \longleftarrow$ apply $\mathsf{d}\mathcal{L}$ proof rule to g
        $G \longleftarrow G \cup \{\tilde{g}\}$
  $\chi_\text{m} \longleftarrow \bigwedge_{s\in S} s$ `// Collect open sequents`

E Simulation

To illustrate the behavior of the water tank model with a fallback controller,we created two monitors: Monitor χ_m validates the complete model (as in the examples throughout this paper) and is executed at the beginning of each control cycle (before the controller runs). Monitor χ_c validates only the controller of the model α (compares prior and post state of $f := *;\ ?-1 \leq f \leq \frac{m-x}{\varepsilon}$) and is executed after the controller but before control actions are issued. Thus, monitor χ_c resembles conventional runtime verification approaches, which do not check CPS behavior for compliance with the complete hybrid model. This way, we detect unexpected deviations from the model at the beginning of each control cycle, while we detect unsafe control actions immediately before they are taken. With only monitor χ_m in place we would require an additional control cycle to detect unsafe control actions[8], whereas with only monitor χ_c in place we would miss deviations from the

[7] Usually, we want a monitor for all the bound variables of the hybrid system model, i. e., $\mathcal{V} = BV(\alpha)$. [8] We could run monitor χ_m in place of χ_c to achieve the same effect. But monitor χ_m implements a more complicated formula, which is unnecessary when only the controller output should be validated.

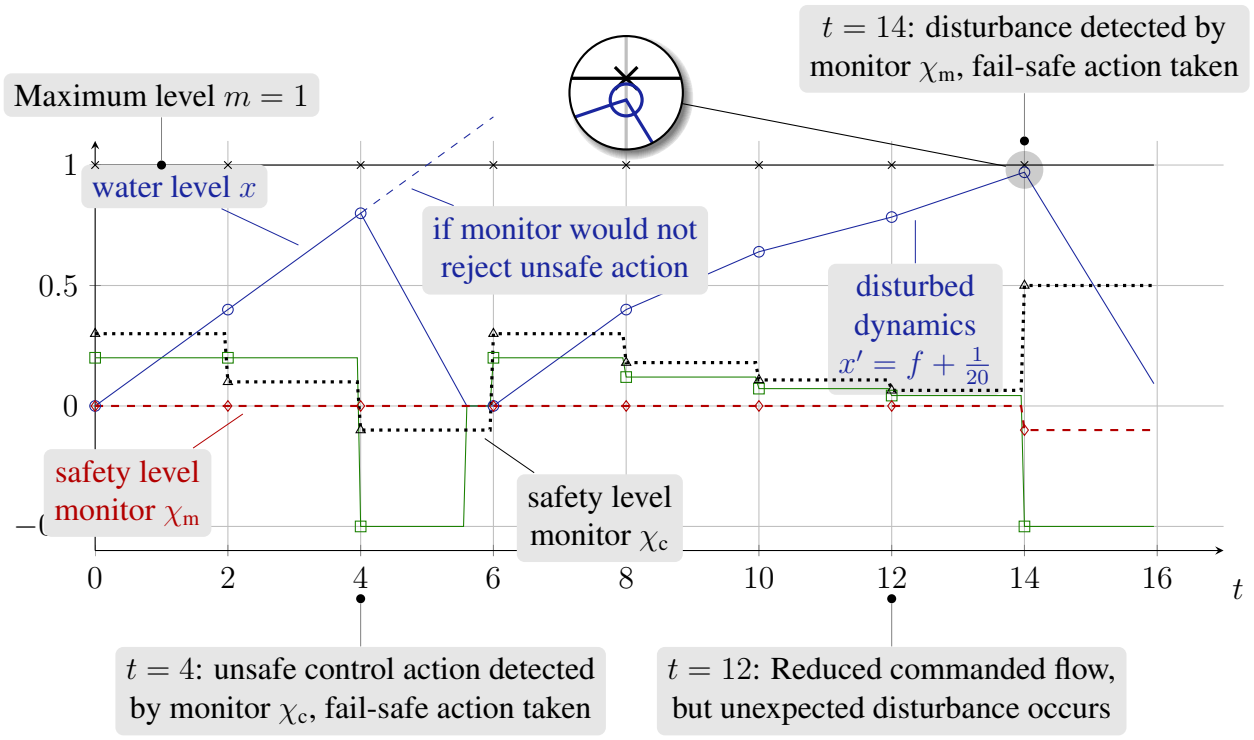

Figure 2: Water tank simulation with monitor illustration, —×— is maximum level (m), —○— is current level (x), —□— is commanded flow (f), - -◇- - is the output of monitor χ_m for the complete model, and ··△·· is the output of monitor χ_c for the controller

model.

Fig. 2 shows a plot of the variable traces of one simulation run. In the simulation, we ran the pump controller every 2 s ($\varepsilon = 2$ s, indicated by the grid for the abscissa and the marks on sensor and actuator plots). The controller was set to pump with $\frac{5(m-x_0)}{2\varepsilon} = \frac{5}{2}$ for the first three controller cycles, which is unsafe on the third controller cycle. Monitor B immediately detects this violation at $t = 4$, because on the third controller cycle setting $f = \frac{5}{2}$ violates $f \leq \frac{m-x_1}{\varepsilon}$. The fail-safe action at $t = 4$ drains the tank and, after that, normal operation continues until $t = 12$. Unexpected disturbance $x' = f + \frac{1}{20}$ occurs throughout $t = [12, 14]$, which is detected by monitor χ_m. Note, that such a deviation would remain undetected with conventional approaches (monitor χ_c is completely unaware of the deviation). In this simulation run, the disturbance is small enough to let the fail-safe action at $t = 14$ keep the water tank in a safe state.